ACADEMIC PRESS

Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Journal of Computer and System Sciences 67 (2003) 46–62

JOURNAL OF COMPUTER AND SYSTEM SCIENCES

http://www.elsevier.com/locate/jcss

On parallel attribute-efficient learning ☆

Peter Damaschke

Computing Science, Chalmers University, Göteborg 41296, Sweden

Received 14 June 2000; revised 13 August 2002

Abstract

This paper continues our earlier work on (non)adaptive attribute-efficient learning. We consider exact learning of Boolean functions of n variables by membership queries, assuming that at most r variables are relevant. The learner works in consecutive rounds, such that the set of simultaneous queries in every round may depend on all information gained so far. For deterministic learning of specific monotone functions we prove that any strategy that uses an optimal query number needs $\Theta(r)$ rounds in the worst case. Furthermore, we make some progress regarding the constant factors in nearly query-optimal strategies. For example, we propose a strategy using roughly $2^{r+1} + 2\eta \log_2 n$ queries in $3r$ rounds. In contrast to the limitations of deterministic strategies, there is a randomized strategy that learns monotone functions by $2^{O}(r) + O(r \log n)$ expected queries in $O(\log r)$ expected rounds. Actually, this result holds in more general function classes. The second part of the paper addresses the computational complexity of parallel learning of arbitrary Boolean functions with r relevant variables. We obtain several strategies which use a constant number of rounds, $O(2^r\, poly(r \log n))$ queries, and only $2^{O(r)}\, n\, poly(\log n)$ computations.

Keywords: Learning by queries; Monotone Boolean functions; Relevant variables; Limits of parallelization; Binary codes; Randomization; Auxiliary computation; Special assignment families

1. Introduction

1.1. Learning model and objectives

We consider exact learning of Boolean functions by membership queries. A Boolean function f of n variables is given as an oracle (“black box”), and a learner wants to identify f exactly. To this

☆This is a merger and revised version of two conference papers presented at the *9th* International Conference on Algorithmic Learning Theory ALT’98, Otzenhausen, Germany, and at the *7th Scandinavian* Workshop on Algorithm Theory SWAT’2000, Bergen, Norway. Preliminary versions appeared in *Lecture* Notes in Computer Science (Springer), Vols. 1501 and 1851.

E-mail address: ptr@chalmers.se.

0022-0000/03/$ - see front matter
doi:10.1016/S0022-0000(03)00047-3

end the learner may ask membership queries. A membership query is the following primitive: The learner chooses an assignment x, assigning a Boolean value 0 or 1 to each variable. Then x is submitted to the oracle which outputs the value $f(x)$.

The learning process consists of a sequence of rounds, and each round in turn consists of the following steps:

ASK. The learner chooses a set of assignments and sends it to the oracle. The oracle answers all these membership queries in parallel.
THINK. The learner receives these oracle responses and performs some auxiliary computations.

In the THINK step of the final round, the learner has to declare that he has learned f, and he has to output some representation of f. In all other rounds, he computes a set of queries to be asked in the next round. In every THINK step, the learner can make use of all information gained so far, i.e. the function values observed in all earlier rounds. A learning strategy is called deterministic if THINK depends on these oracle answers only, whereas in a randomized strategy the result of THINK may also depend on random bits available to the learner.

The number of rounds may or may not be fixed in advance. A strategy running in only one round is called nonadaptive. The other extreme case is the class of adaptive strategies where only one query can be asked in every round. Note that nonadaptive learning of arbitrary functions from a given function class is equivalent to the problem of choosing a query family that uniquely determines every function from this class. That is, different functions must yield different vectors of oracle answers. (However practical strategies must satisfy a further demand, see below.) In the algorithmic learning theory literature, such a family is also called a universal teaching set for the function class.

Our goal is to learn any given f from a restricted class of Boolean functions. The learning strategy should use a minimum number of queries and a minimum number of rounds. More precisely, we are interested in worst-case bounds in terms of input size parameters. In randomized strategies, we usually want to minimize the expected number of queries and rounds, respectively. A randomized strategy is called a Las Vegas strategy if the final result is always correct, whereas the output of a Monte Carlo strategy must be correct only with high probability.

The query number somehow measures the total cost of learning while the number of rounds corresponds to the parallel execution time. Another important issue besides the pure query complexity is the amount of auxiliary computations in the THINK steps which should not be unreasonably high.

1.2. Attribute-efficient learning

Clearly, all 2^n possible queries must be asked if nothing about f is known in advance. However if the learner was promised that f belongs to some restricted class of Boolean functions, clever query strategies can benefit from this prior knowledge.

There are trivial examples of function classes where 2^n queries are still necessary despite this background information, consider e.g. the class of functions that have value 1 for exactly one assignment. Even randomization cannot help in such cases. (Due to some recent fascinating results, quantum computers can solve search problems by surprisingly few queries, subject to

some error probability, see e.g. the survey article [18]. However in the present paper we stick to the classical setting.) On the positive side, various important function classes admit efficient learning, due to structural properties of the functions therein. In the present paper we consider a different type of restriction:

A variable v is called relevant if there exists at least one assignment x such that $f(x) \neq f(x')$, where x' is the assignment obtained from x by switching the value of v. Otherwise v is called irrelevant. Throughout the paper let r denote the number of relevant variables (attributes) of f, or a known upper bound on this number. We are interested in complexity bounds for queries, rounds, etc., in terms of both r and n. The notion of attribute-efficient learning refers to strategies whose complexity is bounded by certain functions of r and n, in such a way that, for every fixed n, learning is strictly more efficient for smaller r than for larger r. (This is not a precise definition but only an informal explanation, as the bounds depend on the particular problem.)

1.3. Previous work

Various aspects of attribute-efficient learning have been studied e.g. in [3,4,9,12,13,15,17,19,20]. This list is not exhaustive. Known results include a general theory, tight bounds for specific function classes, attribute efficiency in other learning models, and more. An important special case of attribute-efficient learning is group testing, where the learner knows in advance that f is the disjunction of the relevant variables, see e.g. [5,7,8,11,14]. Group testing as well as attribute-efficient learning in general has interesting applications in fields like planning of chemical and biological test series, error detection in hardware and software, and pattern recognition. We refer to the pointers in the quoted papers for further information. Parallelity is essential in applications where the tests (queries) are time consuming but simultaneously executable procedures [1,2,10].

Learning monotone Boolean functions is placed somewhere between group testing and learning arbitrary Boolean functions. Recall that a Boolean function f is called monotone if $x_i \leqslant y_i$ for all i implies $f(x_1, \ldots, x_n) \leqslant f(y_1, \ldots, y_n)$.

The present work directly continues our earlier paper [6] in which we presented two main results:

(i) a strategy that learns monotone functions with r relevant variables using a total of $O(2^r + r \log n)$ queries in $O(r)$ rounds, without previous knowledge of r,
(ii) a strategy that learns arbitrary functions with r relevant variables using $O(r^2 2^r + r 2^r \log n)$ nonadaptive queries. Here, of course, the number r (or an upper bound on r) must be known in advance.

The query bound in (i) is optimal for trivial information-theoretic reasons, and for the problem in (ii), an $\Omega(2^r \log n)$ lower bound is known even for adaptive strategies. While the strategy in (i) is computationally simple, our nonadaptive strategy in (ii) for arbitrary Boolean functions does not immediately lead to an efficient method that computes a representation of output f (that is, the set of relevant variables and a truth-table) from the oracle responses. This is a crucial problem if someone wishes to use such a strategy in a practical application.

In the present paper we show that result (i) is asymptotically tight, and we study the constant factors. Furthermore, we propose computationally efficient strategies that overcome the mentioned drawback of the query-optimal strategy in (ii). We use the results from [6], but we do not assume familiarity with that paper. All the necessary prerequisites will be introduced here.

1.4. New results and overview of the paper

In Section 3.1 we prove that any deterministic strategy that learns monotone functions with r relevant variables needs $\Theta(r)$ rounds, if we insist on an optimal query number, which is $O(r \log n)$. More generally, there is an exponential trade-off: A k-round strategy needs $\Theta(k2^{r/k} \log n)$ queries. This result is even true for some specific monotone Boolean function whose structure may be revealed. Only the location of the relevant variables must be kept secret. Our proof is an adversary argument using an inequality for minimum Hamming distances in binary codes (Lemma 1).

Due to this lower bound result, it makes sense to study the constant factors in nearly query-optimal strategies. This will be done in Section 3.2. We propose a strategy that uses roughly $2^{r+1} + 2r \log_2 n$ queries in $3r$ rounds. The skeleton of this strategy is taken from [6]; however, we have to add several new ideas, because nonobvious refinements of the original strategy and its analysis are necessary to obtain these coefficients. A second strategy that we will propose runs in only $2r$ rounds, but it needs the threefold number of queries.

The lower bound from Section 3.1 applies to deterministic strategies. In contrast to this result, there is a randomized strategy that learns monotone functions by $2^{O(r)} + O(r \log n)$ expected queries in only $O(\log r)$ expected rounds, as we will show in Section 3.3. Actually, this remains true in more general function classes. The strategy uses the doubling technique, ideas from hashing, and result (ii) mentioned above. (Here we remark that we developed a randomized solution to another generalization of group testing which provably lacks an efficient deterministic strategy in [5].)

These techniques are also applied in Section 4, in order to reduce the computational complexity of parallel learning of arbitrary Boolean functions with r relevant variables. A naive method would require more than n^r computations (see Lemma 5), which is barely practical. Instead of the naive use of (ii) we will obtain several strategies that require a larger but still constant number of rounds, $O(2^r poly(r \log n))$ queries, and only $r^{O(r)} n\, poly(\log n)$ computations. (Complexity bounds

Table 1

Rounds	Queries	Operations	Type
1	$O(r^4 2^r \log r \log^2 n)$	$2^{O(r)} n \log^2 n$	Deterministic
2	$O(r^3 2^r \log n)$	$r^{O(r)} n \log n$	Deterministic
2	$O(r^2 2^r + r \log n)(E)$	$r^{O(r)} n \log n(E)$	Monte Carlo
$O(1)(E)$	$O(r^2 2^r + r 2^r \log n)(E)$	$r^{O(r)} n \log n(E)$	Las Vegas
5	$O(r^2 2^r \log r + r 2^r \log n)(E)$	$r^{O(r)} n \log n(E)$	Las Vegas

refer to the standard RAM model of computation.) The exponential term in r is inevitable, but for fixed (and typically small) r, our complexity bound is $O(n\,poly(\log n))$.

Table 1 summarizes these results. Bounds with label (E) are expected values, the others are worst-case bounds.

The results are based on Theorem 2 from [6] and various lemmas provided in Section 2. We do not claim that our strategies achieve the ultimate complexity bounds, but at least they make parallel attribute-efficient learning more accessible in practice.

2. Preliminaries

2.1. Notation

We assume that the reader is familiar with Boolean functions. Therefore, we introduce only notation which has a specific meaning in this paper.

Let V denote the set of variables of the Boolean function f to be learned. An assignment x on V is a mapping of V into the set of Boolean values $\{0,1\}$. An assignment on V induces an assignment on any subset of V in an obvious sense. Sometimes we identify an assignment x with the set $W\subseteq V$ of those variables that have value 1 in x. According to this abuse of notation, we sometimes write $f(W)$ instead of $f(x)$. The phrase "to query a set W" means to ask the function oracle about the value $f(W)$.

The notion of relevant variables has already been defined in Section 1.2.

For $X\subset V$ and an assignment y on $V\backslash X$, the projection f_y is the Boolean function on X having the same values as f when this particular y remains fixed.

Consider a partition π of V into disjoint subsets, called bins. Empty bins are allowed. The coarsening f_π of f is the function defined as follows: The variables of f_π are the bins, and for any assignment of Boolean values on the set of bins, called a bin assignment, the value of f_π equals the value of f, where each variable inherits the value assigned to the bin it belongs to. Note we can use an oracle for f as an oracle for f_π as well. Relevant variables of f_π are also called relevant bins. Clearly, a relevant bin contains at least one relevant variable of f, but the converse is not true in general.

2.2. Some lemmas

In this section we collect several technical prerequisites, in order to streamline the presentation of our main results in Sections 3 and 4.

Many learning problems for Boolean functions are closely related to binary codes and combinatorial designs. The Hamming distance of two bit vectors (or assignments) is the number of positions where these vectors disagree. It is a straightforward exercise to prove: Among any three binary words of equal length q, there exist two words with Hamming distance at most $(2/3)q$. However, a stronger estimate is known:

Lemma 1. *The minimum Hamming distance d among n binary words of length q satisfied $d\leqslant(qn)/2(n-1)$.*

This is called the Plotkin bound, see [16] or any textbook on error-correcting codes. For large n we may pretend that $d \leqslant q/2$: The error caused by this simplification becomes arbitrarily small if $n \to \infty$.

One of the main results of [6] was:

Theorem 2. *Arbitrary Boolean functions with at most r relevant variables can be learned by $O(r^2 2^r + r2^r \log n)$ queries in one round, if r is known in advance. Moreover, a random family of $O(r^2 2^r + r2^r \log n)$ assignments is, with high probability, a universal teaching set for the class of functions with at most r relevant variables.*

The proof is involved, and the explicit construction of such query families is apparently a difficult matter. However case $r = 1$ is trivial (the obvious proof is left to the reader):

Lemma 3. *Boolean functions with only one relevant variable can be learned by $O(\log n)$ queries in one round, and $O(n \log n)$ auxiliary computations.*

Actually, these "computations" are simple assistance operations such as setting the bits in oracle queries, and performing binary search for the relevant variable using the oracle answers. For arbitrary but fixed r, the naive way to reconstruct the given f from the oracle answers requires more than n^r computations. Before we can explain this claim we need some more details from [6]: Let A the family of queries used by our (or any other) nonadaptive learning strategy. A set $U \subset V$ is called (A,f)-feasible if, for all assignments $a \in A$, value $f(a)$ depends only on the assignment that a induces on U. In [6] we have shown:

Lemma 4. *The set R of relevant variables of f is the unique minimum (A,f)-feasible set.*

Now we are ready to prove the following result which limits the computational complexity (not only the query number) of nonadaptive learning:

Lemma 5. *Arbitrary Boolean functions with at most r relevant variables can be learned nonadaptively, by querying an assignment family as in Theorem 2 and $O(n^r \log n)$ subsequent auxiliary computations.*

Proof. Just check all sets $R \subset V$ of size r for (A,f)-feasibility. The number of these subsets is approximately $n^r/r!$. Every R can be checked in $O(r|A|)$ time. This follows immediately from the definition of (A,f)-feasible sets: Simply sort the assignments induced on R lexicographically. Now observe that $|A| = O(r^2 2^r \log n)$ by Theorem 2, and that $r^3 2^r/r!$ is bounded by some constant. □

Some other special assignment families which we discuss below will serve as building blocks of our strategies in Section 4.

We say that a partition of V into bins separates $R \subset V$ if any two elements of R belong to distinct bins, or equivalently, if every bin contains at most one member of R. A family of

partitions of V with $b \geqslant r$ bins each is called (b,r)-separating if every set $R \subset V$ of size r is separated by at least one partition from this family.

There are slight differences to similar common concepts like shattering families, splitters, and perfect hash functions. To avoid confusion, we have introduced our own definition. Note that we allow bins of different size.

The following lemma is proved by straightforward application of the probabilistic method.

Lemma 6. *There exist (r^2,r)-separating families of size $O(r \log n)$.*

Proof. Throw the elements of V independently and equiprobably into one of the b bins. Consider any set $R \subset V$ of size r. Obviously, the probability that R is separated is at least $(1-r/b)^r \approx e^{-r^2/b}$. Hence the probability that some r-element subset remains unseparated by t random partitions is less than $n^r(1-e^{-r^2/b})^t$. Choosing $t = \Theta(e^{r^2/b} r \log n)$ with large enough constant factor, we can makes this probability bound smaller than 1, thus there must exist a counterexample. Finally let be $b = r^2$. □

A family A of assignments on a set V of n Boolean variables is (n,r)-universal if, on each subset of r variables, each of the 2^r possible assignments is induced by some member of A. By convention, any nonempty A is at least $(n,0)$-universal. The upper bound $O(r2^r \log n)$ on the cardinality of optimal (n,r)-universal families is well known. (Once more the probabilistic method provides a simple existence proof.) In particular, note that the families addressed in Theorem 2 must be (n,r)-universal.

Let R be the set of relevant variables of f, and assume that some bound $r \geqslant |R|$ is known. Under this assumption, universal families can be used to check in one round whether a given $S \subseteq R$ with s elements equals R:

Lemma 7. *Let A consist of all $O(r2^r \log n)$ possible pairs of an arbitrary assignment on S and an assignment from some $(n-s, r-s)$-universal family on $V \backslash S$. Then we have $R \backslash S \neq \emptyset$ if and only if S is not (A,f)-feasible.*

Proof. The "if" direction is trivial. To prove the "only if" part, assume that $V \backslash S$ contains (at most $r-s$) relevant variables. For any $v \in R \backslash S$ there exists an assignment b on R such that f changes its value if we switch b on v only. Since we have coupled arbitrary assignments on S and an $(n-s, r-s)$-universal family on $V \backslash S$, some $a \in A$ induces b on R. Hence S is not (A,f)-feasible, contradiction. □

3. Monotone functions

3.1. Lower bound for queries vs. rounds

For proving a lower bound trade-off we first show that the power of nonadaptive queries to detect relevant variables of a monotone function is severely limited. For convenience let be $\log_2 s = 0$ if $s < 1$.

Let R with $|R| = r$ be the set of relevant variables of f. Assume that the learner has already obtained $f(x)$ for all x from some set Q of queries. A set $U \subset V$ is said to be consistent with these function values if there exists a function g such that $g(x) = f(x)$ for all $x \in Q$, and the set of relevant variables of g is exactly U. We define the uncertainty u as the largest number for which there exists a subset $S \subseteq R$, $|S| = r - u$, such that every $U \supseteq S$ of size r is consistent. Intuitively this means that the learner does not know anything about the u relevant variables outside S.

Lemma 8. *There exists an adversary strategy achieving that, in every round with $s \log_2 n$ queries, the learner can reduce his uncertainty by at most $2 + b \log_2 s$, where $\lim_{n \to \infty} b = 1$.*

Proof. The adversary presents the monotone Boolean function f described in the following. The r relevant variables of f are denoted by $x_1, x_2, \ldots, x_r$, and the DNF of f contains all terms consisting of one variable x_j, j odd, along with all $x_i, i<j$, i even. In other words, f can be written as $x_1 \vee x_2x_3 \vee x_2x_4x_5 \vee x_2x_4x_6x_7 \vee x_2x_4x_6x_8x_9 \vee \cdots$.

The adversary may even betray the fact that f has this shape, but he does not tell the learner what the relevant variables are.

We observe some crucial properties of this function:

- $x_1 = 1$ implies $f = 1$.
- $x_1 = x_2 = 0$ implies $f = 0$.

By the self-similarity in the structure of f, this recurs for all odd j, in the following sense. Fix $x_i = 0$ for all odd $i<j$, and $x_i = 1$ for all even $i<j$. Then we have

- $x_j = 1$ implies $f = 1$.
- $x_j = x_{j+1} = 0$ implies $f = 0$.

Assume that the learner asks $s \log_2 n$ queries in the first round. The query set can be considered as a binary code of length $q = s \log_2 n$ that assigns a code word to each variable. The adversary exploits the Plotkin bound and picks two words (i.e. variables) y, z which differ in at most $\frac{s}{2} \log_2 n$ bits, subject to an excess that can be neglected if n is large enough. The set of queries submitted in the first round can be partitioned into three subsets, with respect to the values assigned to y and z:

- X, containing the queries where $y = z$,
- Y, containing the queries where $y = 1$, $z = 0$,
- Z, containing the queries where $y = 0$, $z = 1$.

If $|Y| > |Z|$ then the adversary fixes $x_1 = y$, $x_2 = z$. If $|Y| \leqslant |Z|$ then the adversary decides that $x_1 = z$, $x_2 = y$. Since $|Y + Z| \leqslant \frac{s}{2} \log_2 n$, set Y or Z (the one with smaller cardinality) contains at most $\frac{s}{4} \log_2 n$ queries. One should notice that the whole argument relies on the assumption that queries are nonadaptive.

Next, we describe how the adversary responds to all these queries. Each query with $x_1 = 1$ is answered $f = 1$. If $x_1 = x_2 = 0$ then the adversary outputs $f = 0$. The point is that these queries do not provide any information about the remaining relevant variables: No matter which of the variables are the x_i, $i \geqslant 3$, we have consistency with the answers given by the adversary. In other

words, the learner can only exploit the remaining queries, i.e. those with $x_1 = 0$ and $x_2 = 1$, in order to recognize further relevant variables. Due to the choice of x_1, x_2 these are at most $\frac{s}{4}\log_2 n$ queries.

The adversary considers this query set (Y or Z) again as a binary code with n words of length reduced by factor 4. He chooses x_3 and x_4 similarly as above, and so on. Then the argument can be repeated, and after $\log_4 s$ steps there remains a code of length $\log_2 n$ or less. At this stage the adversary may even choose two of the n variables which are not distinguished by the remaining code words. We conclude that, in the first round, the learner is able to identify only the first $2 + 2\log_4 s = 2 + \log_2 s$ relevant variables, without getting further information on the other relevant variables.

Finally, this argument also applies to the following rounds, where the earliest x_j not yet recognized plays the role of x_1. □

Now we can derive the following lower bound. For simplicity we pretend again that $b = 1$ in Lemma 8. The error vanishes with growing n.

Theorem 9. *Any deterministic strategy that learns monotone Boolean functions with r relevant variables in k rounds, with k bounded by some fixed fraction of r, needs* $\Theta(k2^{r/k}\log n)$ *queries in the worst case.*

Proof. Let r_i be the decrease of uncertainty in the ith round, provided that the adversary observes the strategy of Lemma 8. First consider the case that $r_i > 2$ for all i. Then it follows from the lemma that more than $2^{r_i-4}\log_2 n$ queries have been asked in the ith round. Since we have $\sum_i r_i = r$, and the function 2^{x-4} is convex from below, the lower bound on the total query number $\sum_i 2^{r_i-4}\log_2 n$ is minimized if $r_i = r/k$ for all i. From this observation the theorem follows in that case.

Now consider the general case that $r_i = 2$ in $m \leqslant k$ rounds, and $r_i > 2$ in the remaining $k - m$ rounds. (Note that our adversary unveils relevant variables pairwise.) If $k < r/2$ then $m < k$. By the same convexity argument as above, it suffices to minimize $(k-m)2^{(r-2m)/(k-m)-4}$. If we differentiate this term by m, we obtain $\ln 2(r-2k)/(k-m) - 1$, subject to some positive constant factor. This derivative is positive for $k < r/4$, hence our function is monotone increasing with m. Thus it is minimized if $m = 0$. □

Corollary 10. *Any deterministic strategy using the asymptotically optimum query number* $O(r\log n)$ *can be forced to spend* $\Omega(r)$ *rounds.*

That means, our strategy from [6] has optimal parallelity, up to constant factors.

3.2. Coefficients in nearly query-optimal deterministic strategies

In view of Corollary 10, it is now interesting to study the constant factors t, p, q in deterministic strategies that use $t2^r + qr\log_2 n$ queries in pr rounds. (In [6] we did not consider constant factors.) Coefficient t is easy to settle by the next lemma which implies that $t = 2$ is optimal.

Before we state the lemma, we introduce the notion of a termination test. Assume that the learner gets the set R of relevant variables for free. To verify this alleged set of relevant variables, the learner has to check that there are in fact no relevant variables outside R. We define the termination test for R as the following set of 2^{r+1} queries: For each assignment on R, ask a pair of queries where all variables outside R are 0 or 1, respectively. If, for every such pair of queries, the two values of f are equal, then no relevant variables outside R can exist. This follows immediately from monotonicity of f. On the other hand we have:

Lemma 11. *Any strategy that learns monotone Boolean functions with r relevant variables must perform the termination test for the set R of relevant variables. Consequently it needs at least* 2^{r+1} *queries.*

Proof. Assume that some query x from the termination test is missing, w.l.o.g. let x be 1 on all variables outside R. Then, if all queries y with $y \geqslant x$ yield $f(y) = 1$, and all other queries y yield $f(y) = 0$, it is impossible to decide whether $f(x) = 0$ or $f(x) = 1$. □

In contrast to this result for t, it is rather difficult to obtain optimal results for factors p and q. We have the trivial information-theoretic lower bound $q \geqslant 1$, but it arises the question how small we can actually make q. The following strategy—the best one with respect to q we could find—achieves $q = 2 + o(1)$ and needs $p = 3$.

Theorem 12. *Monotone Boolean functions with at most r relevant variables can be learned by* $2^{r+1} + (2 + o(1))r \log_2 n$ *queries in* $3r$ *rounds.*

Proof. Let f be the given function, and let V be the set of Boolean variables of f. Assume $f(\emptyset) = 0$ and $f(V) = 1$, otherwise f is constant, and we are done.

We put the variables on distinct vertices of a k-hypercube of dimension $k = \lceil \log_2 n \rceil$. Some irrelevant dummy variables may be added if n is not a power of 2. This can at most double the number of variables and does not affect the coefficients. A k-hypercube defines, in a natural way, k pairs of $(k-1)$-hypercubes. In the first round we query k of these $(k-1)$-hypercubes, exactly one from each pair. Consider a $(k-1)$-hypercube W that has been queried. If $f(W) = 0$ then W is called a lower set, and $V \backslash W$ is called an upper set. Similarly, if $f(W) = 1$ then W is called an upper set, and $V \backslash W$ is called a lower set.

Next, we perform two rounds with k assignments formed in the following way (but only a selection of them will be queried, as specified below): Arrange the upper sets in arbitrary order. Then the candidates for query sets are the intersections of the first i upper sets, for $i = 1, \ldots, k$. We partition this chain of nested sets into approximately $\sqrt{k}$ segments of length about $\sqrt{k}$. In the second round, we query the first set (containing a single variable v) and the $\sqrt{k}$ sets that bound the segments. If $f(\{v\}) = 1$ then v is relevant. Thus we have found some relevant variable after 2 rounds with $k + \sqrt{k}$ queries.

In the other case $f(\{v\}) = 0$, there must be a jump from $f = 0$ to 1 in our chain. In the third round we query the $\sqrt{k}$ sets of the segment where this jump occurs. Let W and $W \cup W'$ be those neighbored sets in our chain which satisfy $f(W) = 0$ and $f(W \cup W') = 1$. Due to the construction,

there is a uniquely determined pair of $(k-1)$-hypercubes H and H' satisfying the following: $W \subset H$, $W' \subset H'$, H is an upper set, and H' is a lower set.

Now there might be two subcases. If $f(H) = f(H')$, one easily sees that both H and H' contain at least one relevant variable. (Since f is monotone, this holds for any complementary pair of sets where f has the same value.) If $f(H) \neq f(H')$ then obviously $f(H') = 0$. Since f is monotone, this implies $f(W') = 0 = f(W)$. Together with $f(W \cup W') = 1$ we conclude, similarly as above, that both W and W' contain relevant variables.

In order to being able to tell the difference between these subcases, the learner has to query both H and H'. But in the first round only one of H and H' has been queried. The second query might be asked in a fourth round, however we can save this round by preventively asking all these partner queries in the interesting segment already in the third round. These are $\sqrt{k}$ additional queries. In summary, we have found two disjoint subsets containing relevant variables after 3 rounds using a total of $k + 3\sqrt{k}$ queries. This situation is called a splitting.

The whole procedure described above is recursively applied to the mentioned subsets X that contain relevant variables. In every such subprocedure, a suitable assignment y on $V \backslash X$ remains fixed: Use any y such that the projection f_y is not constant on X. Such an assignment y is immediately obtained from the splitting. (Details should be clear from the case distinction above.)

This recursive application yields a binary tree whose nodes are subsets of variables. Every leaf contains one relevant variable. If no further splitting can be made then the tree is ready, and we test whether all relevant variables already appear in the leaves, using the termination test introduced prior to Lemma 11. In the negative case we find some nonconstant projection of f, and we can continue the process constructing a new tree, and so on. The number of queries in all termination tests is 2^{r+1}, although we perform termination tests for several sets $S \subseteq R$ (and only the last one is affirmative). To see this, note that we run termination tests in a growing sequence of query sets $S \subseteq R$. Since repetition of earlier queries is needless, we actually do the termination test for R only.

Altogether we have obtained a sequence of binary trees with the meaning described above. Since every inner node has two children, the total number of inner nodes in all our trees is at most $r-1$, and the sum of depths of all trees is at most r. Remember that an inner node (corresponding to a splitting) represents 3 rounds, whereas a leaf represents 2 rounds. Since every termination test adds one further round to the deepest leaves in every tree, we may pretend that every node represents 3 rounds. This gives $p = 3$ in the worst case. The number of queries (excluding those in the termination tests) is bounded by

$$r(k + \sqrt{k}) + (r-1)(k + 3\sqrt{k}) = (2r-1)k + (4r-3)\sqrt{k} = (2 + o(1))k.$$

We remark that $q \leqslant 2$ for $\log n > 16r^2$. □

One may also try to minimize p while keeping q constant. The best result we have found in this direction is: $p = 2$, $q = 6$, and $t = 2r$. (Here t is no longer constant, but still independent of n.)

Theorem 13. *Monotone Boolean functions with at most r relevant variables can be learned by* $2r \cdot 2^r + 6r \log_2 n$ *queries in* $2r$ *rounds.*

Proof. Our strategy has the same overall structure as in Theorem 12, and the details are much simpler. Therefore we only discuss the differences to the previous strategy. Every tree node represents 2 rounds. In the first round we query all $2k$ hypercubes of dimension $k-1$. In the second round we query all sets in the chain of intersections of the first i upper sets, for $i = 1, \ldots, k$. Remember that the first set in the chain contains a single variable v. Simultaneously, i.e. still in the second round, we run the termination test for the set T which contains all previously detected relevant variables and, in addition to them, all variables v from the tree nodes, as mentioned above. The tree nodes are processed in parallel. These variables v are only suspected to be relevant, however T contains at most r elements, since the number of nodes in each level of the current tree is bounded by r minus the number of leaves in earlier levels. Thus the termination test on each level consists of at most 2^{r+1} queries. Since the depth of all trees is bounded by r, we get $t = 2r$ and $p = 2$. Moreover, every tree node stands for $3k$ queries, and the number of nodes is less than $2r$, thus $q = 6$. □

Finally, we mention that Theorem 9 immediately yields an exponential lower bound on the p vs. q trade-off:

Corollary 14. *If a deterministic strategy learns monotone Boolean functions with r relevant variables in pr rounds using $t2^r + qr\log_2 n$ queries, with t independent of n, then $q = \Theta(p2^{1/p})$.*

3.3. Randomization helps

In this section we show that a randomized strategy can learn monotone functions by roughly $O(r\log n)$ queries in $O(\log r)$ rounds. This contrasts nicely to the deterministic case. Moreover note that the learner is not supposed to have any prior knowledge about r.

Theorem 15. *There is a Las Vegas strategy that learns monotone Boolean functions with r relevant variables using an expected number of $2^{O(r)} + O(r\log n)$ queries in an expected number of $O(\log r)$ rounds.*

Proof. For $s = 1, 2, 4, 8, 16, \ldots$, perform 3 rounds as described below, until the termination criterion in (3) is fulfilled:

(1) Throw the n variables at random into 2^{3s} bins. Then consider the coarsening f_π with respect to this random partition π, and apply the strategy from Theorem 2 to learn up to s relevant bins by $O(s^2 2^s)$ queries. Note that this step fails if $s < r$ but f_π has more than s relevant bins, however we will catch such failures in (3).

(2) Search every relevant bin for one relevant variable. In every bin with exactly one relevant variable, this can be done by $\log_2 n$ queries in one round by Lemma 3, and we can search all these bins in parallel. (Details need some care, but they are straightforward.) This is a total of at most $s\log_2 n$ queries. Note that this step can also fail if some bins contain several relevant variables.

(3) Check whether all relevant variables have been found, using the termination test from Lemma 11 that needs at most 2^{s+1} queries. If the test is negative, double s and go to (1).

Although the first rounds may fail for the reasons mentioned in (1) and (2), step (3) ensures correctness of the final outcome. Triples of rounds performed after having reached $s \geqslant r$ are called trials in the following. As soon as the r relevant variables get into r different bins in some trial, all relevant bins will be found in step (1), and all the relevant variables will be found in step (2), which is verified afterwards in step (3). Hence a failure in a trial can appear only if the relevant variables are not in r distinct bins. Therefore the failure probability is at most $r^2/2^{3s}$.

The first trial needs $2^{O(r)}$ queries in steps (1) and (3), and $O(r \log n)$ queries in step (2). Since s has always been doubled, the total query number in all previous rounds (with $s < r$) is within these bounds. If a trial with parameter s fails then the next trial with parameter $2s$ submits $O(4s^2 2^{2s})$ queries in steps (1) and (3), and $2s \log_2 n$ queries in step (2). However, this trial is performed only with probability less than $r^2/2^{3s}$. (Actually this is a generous bound.) That means, the expected query number of the next trial is bounded by $O(r^2 s^2/2^s + r^2 s \log n/2^{3s})$. The sum of these terms over all trials is convergent, hence the first trial dominates the expected query number. The expected number of rounds is obviously $3 \log_2 r + O(1)$. □

We conclude this section with a list of comments:

- If r is known in advance then, obviously, the same query number as in Theorem 15 can be achieved in $O(1)$ rounds.
- We have used 2^{3s} bins for ease of presentation, but a smaller number would be sufficient. Moreover, we may abort the process as soon as the prospective number of bins exceeds n: Then we may simply consider f instead of a coarsening. However these improvements do not reduce the complexity bounds.
- We only considered the expected complexity. It is a drawback of our strategy that the query number grows dramatically if the first trial fails. This is not very likely but inconvenient if it happens nevertheless. A strategy with smaller variance in the query number is desirable.
- Note that monotonicity is not really exploited: We used monotonicity only for the termination test in step (3). Therefore the same strategy is applicable to either class of Boolean functions enjoying the following properties: The class is closed under projection, and there is a test with $O(1)$ queries that decides whether a given function from the class is constant.
- The use of Theorem 2 is essential to our strategy. We do not see how this could be avoided.
- Since our strategy uses a large amount of randomness anyway, we may use random assignments in step (1), rather than explicitly constructed families (cf. Theorem 2). Although this adds a further source of failure, the strategy remains Las Vegas because of step (3).
- Since we apply Theorem 2, the complexity of computing f_π and the relevant bins from the oracle answers becomes an important issue. This leads us to the next part of the paper.

4. Arbitrary functions: saving computations

We will now propose several parallel strategies for learning arbitrary Boolean functions f, provided that at most r of the n variables are relevant, and r is known in advance. These strategies are fairly efficient regarding both the query number and the computational complexity. Our bounds are pairwise incomparable, hence each of these strategies can be the best choice under

certain circumstances. It should be noticed that we only count the computations being necessary to process the oracle answers during the learning procedure given an instance f, but not the time needed to construct the special query families the strategies require. This is appropriate, since these families depend on n and r only and must be constructed only once for any size. Once the families are stored, they can be used for all instances f of that order of magnitude. Throughout the section, R denotes the set of relevant variables of f.

4.1. Learning in one round

Theorem 16. *Boolean functions with at most r relevant variables can be learned by $O(r^4 2^r \log r \log^2 n)$ queries in one round, followed by $2^{O(t)} n \log^2 n$ computations.*

Proof. The construction consists of three nested structures.

(1) Take an (r^2, r)-separating family of $O(r \log n)$ partitions, as provided by Lemma 6.

(2) For every π from (1), take an (r^2, r)-universal family of size $O(r2^r \log r)$ on the set of bins of π. This yields a total of $O(r^2 2^r \log r \log n)$ bin assignments for the various partitions π. We may consider them as assignments on V, in an obvious sense.

(3) Every assignment x on V from (2) is replaced with $O(r^2 \log n)$ new assignments in the following way: For every bin U (in the partition π where x comes from), fix the assignment induced by x on $V \backslash U$, and replace the all-0 assignment or the all-1 assignment on U with the members of some $O(\log n)$ family that enables nonadaptive search for one relevant variable, as given by Lemma 3.

The learning algorithm queries all assignments produced above. We prove that this is sufficient to uniquely identify any Boolean function with r relevant variables.

Consider any bin U. Whenever a family of $O(\log n)$ assignments from step (3) finds some relevant variable of some projection of f to U then, trivially, this is also a relevant variable of f. It remains to show that the whole set R is found in this way. The argument is straightforward:

Among our partitions there is some π that separates R. For every relevant bin U in this partition π, there exists a bin assignment b on the set of relevant bins such that f_π switches its value if we change b on U only. Since we took an (r^2, r)-universal family of bin assignments, b is induced by one of the bin assignments a chosen in (2). In a subset of queries formed in (3), we fixed the values of a on all bins except U, and we invoked a search for one relevant variable in U. Now it is obvious that the unique relevant variable in U is detected by this subset of queries.

Note that our query family is also (n, r)-universal. Since we have identified R, we have also learned f as a truth table. The bound on the amount of computation follows immediately from Lemma 3. □

Note that we did not use Lemma 5 here. However this blows up the query number, since many search routines are doomed to fail, namely in bins which do not contain exactly one relevant variable. The extra $r^3 \log r \log n$ factor can be significant if queries are expensive. Therefore it is nice that we can get rid of this term if we allow for two rounds. Now we have to exploit Lemma 5, however in a roundabout way.

4.2. Learning in two rounds

Theorem 17. *Boolean functions with at most r relevant variables can be learned using* $O(r^3 2^r \log n)$ *queries in two rounds, and* $r^{O(r)} n \log n$ *computations.*

Proof. Take an (r^2, r)-separating family of $O(r \log n)$ partitions, as provided by Lemma 6. Consider any partition π in this family. Note that f_π has at most r relevant variables (bins). Thus we can learn f_π in one round by $O(r^2 2^r)$ queries and $r^{O(r)}$ computations, due to Lemma 5. For preparing the oracle queries we need only $r^2 2^r n$ computations. This is done for all π in parallel.

Since some π separates R, and the corresponding f_π has the maximum number of relevant variables among all coarsenings of f by partitions from the considered family, we can explicitly find such π by examining the output of the first round. For each of the relevant bins U of π we also find, in a straightforward way, some bin assignment a such that f_π switches its value if we change a on U only. This requires $O(r^3 2^r)$ further computations (lexicographic sorting and comparisons). Now we apply Lemma 3 to search U for the relevant variable, in one additional round. This is done for all U in parallel, which adds $O(r \log n)$ queries and $O(rn \log n)$ computations. □

4.3. Randomized versions

We can further reduce the complexity by randomization. This is worthy of mentioning, since the expected query number becomes competitive with the deterministic bound.

Theorem 18. *Boolean functions with at most r relevant variables can be learned by a Monte Carlo strategy using* $O(r^2 2^r + r \log n)$ *queries in two rounds, and* $r^{O(r)}\, n \log n$ *computations.*

Proof. The trivial case $r = 3$ is settled by Lemma 3, thus we may assume $r > 1$.

Proceed as in Theorem 17, but replace the separating family with a random partition π of V into r^2 bins. From the proof of Lemma 6 we see that π separates R with at least some constant positive probability. Learn f_π by $O(r^2 2^r)$ nonadaptive queries, and in the second round, search each of the (at most r) relevant bins for one relevant variable. We cannot find the entire R only if π did not separate R. Note that a failure may remain undetected, therefore this is only a Monte Carlo strategy. □

Of course, one can arbitrarily reduce the failure probability by running many "copies" of this strategy, but in case of high safety requirements, a Las Vegas strategy is more calming. Finally we propose such a strategy.

Theorem 19. *Boolean functions with at most r relevant variables can be learned by a Las Vegas strategy within the following expected bounds:* $O(r^2 2^r + r 2^r \log n)$ *queries,* $O(1)$ *rounds, and* $r^{O(r)} n \log n$ *computations.*

Proof. The only difference to the previous strategy is that we verify the output, i.e. the alleged set $S \subseteq R$ of relevant variables, in a third round. If we use Lemma 7, this adds $O(r2^r \log n)$ queries. If the test yields $S = R$ then we stop, otherwise we simply repeat the procedure. Since there is a constant positive probability of success, the expected bounds increase by constant factors only. □

A certain modification gives a slightly larger query number, but also a guaranteed fixed number of rounds, which might be interesting for applications with time-consuming queries:

Corollary 20. *There is another Las Vegas strategy using an expected number of* $O(r^2 2^r \log r + r2^r \log n)$ *queries and* $r^{O(r)} n \log n$ *computations, which stops after three rounds with high probability, and after five rounds at the latest.*

Proof. Run $\Theta(\log r)$ copies of the two-round Monte Carlo algorithm in parallel, and accept the maximum output S. Test whether $S = R$ in the third round. If the test is negative, apply the two-round deterministic strategy from Theorem 17. Since the failure probability can be reduced to $O(1/r^2)$, the expected number of further queries can be neglected. □

5. Open problems

- Theorem 3 establishes a lower bound for the queries vs. rounds trade-off. Is this a tight bound? In other words, can we learn monotone Boolean functions by only $k2^{r/k-4} \log_2 n$ queries, for any number k of rounds between 1 and $\Theta(r)$? The difficulty is to recognize proper subsets of relevant variables by a restricted number of nonadaptive queries. The presence of further relevant variables and an unlucky choice of queries may obscure their relevance, cf. [6,8].
- Our proof of Lemma 8 uses DNF with unbounded size and number of terms. Does the lower bound still hold for m-DNF (m-term DNF) with fixed m? In particular, what can be said about parallel group testing? Can the disjunction of r relevant variables be learned by $O(r \log n)$ queries in less than $\Theta(r)$ rounds? (It is known that nonadaptive group testing needs $\Theta(r^2 \log n)$ queries.) What about the conjunction of r relevant variables? Does a lower bound as in Theorem 9 hold for every fixed Boolean function in r variables, if the location of relevant variables in V is kept secret? (That is, the learner knows f subject to permutations of the variables.)
- Is $q = 2$ in Theorem 12 optimal? And is there any positive constant lower bound for p, or can we make p arbitrarily small at cost of q? More specifically, can we achieve $q = O(p2^{1/p})$ (the lower bound from Corollary 14) for any p? It might already be interesting to improve the particular pairs of constants p, q from Section 3.2.
- If we want to learn monotone functions nonadaptively as in Theorem 2, can we extract the relevant variables from the oracle answers easier than in the general case? As for arbitrary functions, is there a more intelligent way than exhaustive search (Lemma 5)? In any case, find improvements upon the results of Section 4.

References

[1] D.J. Balding, D.C. Torney, A comparative survey of non-adaptive pooling designs, in: T. Speed, M.S. Waterman (Eds.), Genetic Mapping and DNA Sequencing, IMA Volumes in Mathematics and its Applications, Vol. 81, Springer, Berlin, 1996, pp. 133–155.
[2] D.J. Balding, D.C. Torney, Optimal pooling designs with error detection, J. Combin. Theory A 74 (1996) 131–140.
[3] A. Blum, L. Hellerstein, N. Littlestone, Learning in the presence of finitely or infinitely many irrelevant attributes, J. Comput. System Sci. 50 (1995) 32–40.
[4] N.H. Bshouty, L. Hellerstein, Attribute-efficient learning in query and mistake-bound models, Ninth Conference on Computational Learning Theory COLT'96, Assoc. Comput. Mach. Press, New York, pp. 235–243.
[5] P. Damaschke, Randomized group testing for mutually obscuring defectives, Inform. Process. Lett. 67 (1998) 131–135.
[6] P. Damaschke, Adaptive versus nonadaptive attribute-efficient learning, Mach. Learning 41 (2000) 197–215.
[7] A. De Bonis, L. Gargano, U. Vaccaro, Group testing with unreliable tests, Inform. Sci. 96 (1997) 1–14.
[8] A. De Bonis, U. Vaccaro, Improved algorithms for group testing with inhibitors, Inform. Process. Lett. 67 (1998) 57–64.
[9] A. Dhagat, L. Hellerstein, PAC learning with irrelevant attributes, 35th IEEE Symposium on Foundations of Computer Science FOCS'99, IEEE Comput. Soc., Los Alamitos, CA, pp. 64–74.
[10] M. Farach, S. Kannan, E. Knill, S. Muthukrishnan, Group testing problems in experimental molecular biology, Compression and Complexity of Sequences'97, IEEE Comput. Soc., Los Alamitos, CA, pp. 357–367.
[11] P. Fischer, N. Klasner, I. Wegener, On the cut-off point for combinatorial group testing, Discrete Appl. Math. 91 (1999) 83–92.
[12] T. Hofmeister, An application of codes to attribute-efficient learning, Fifth European Conference on Computational Learning Theory EuroCOLT'99, Lecture Notes in Artificial Intelligence, Vol. 1572, Springer-Verlag, Berlin, 1999, pp. 101–110.
[13] J. Kivinen, H. Mannila, E. Ukkonen, Learning hierarchical rule sets, Fifth Conference on Computational Learning Theory COLT'92, Assoc. Comput. Mach. Press, New York, pp. 37–44.
[14] E. Knill, Lower bounds for identifying subset members with subset queries, Sixth ACM-SIAM Symposium on Discrete Algorithms SODA'95, Assoc. Comput. Mach. Press, New York, pp. 369–377.
[15] N. Littlestone, Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm, Mach. Learning 2 (1988) 285–318.
[16] M. Plotkin, Binary codes with specified minimum distances, IEEE Trans. Inform. Theory 6 (1960) 445–450.
[17] R.A. Servedio, Computational sample complexity and attribute-efficient learning, J. Comput. System Sci. 60 (2000) 161–178.
[18] A. Ta-Shma, Classical versus quantum communication complexity, SIGACT News 30 (3) (1999) 25–34.
[19] R. Uehara, K. Tsuchida, I. Wegener, Optimal attribute-efficient learning of disjunction, parity, and threshold functions, Third European Conference on Computational Learning Theory EuroCOLT'97, Lecture Notes in Artificial Intelligence, Vol. 1208, Springer-Verlag, Berlin, pp. 171–184.
[20] L.G. Valiant, Projection learning, 11th Conference on Computational Learning Theory COLT'98, Assoc. Comput. Mach. Press, New York, pp. 287–293.